VŠB – Technická univerzita Ostrava
Fakulta elektrotechniky a informatiky
Katedra informatiky

# Absolvování individuální odborné praxe

# Individual Professional Practice in the Company

2017 Jiří Strojil

VŠB - Technická univerzita Ostrava
Fakulta elektrotechniky a informatiky
Katedra informatiky

# Zadání bakalářské práce

Student: **Jiří Strojil**

Studijní program: B2647 Informační a komunikační technologie

Studijní obor: 2612R025 Informatika a výpočetní technika

Téma: Absolvování individuální odborné praxe
Individual Professional Practice in the Company

Jazyk vypracování: čeština

Zásady pro vypracování:

1. Student vykoná individuální praxi ve firmě: SDE - Software Solutions
2. Struktura závěrečné zprávy:
   a) Popis odborného zaměření firmy, u které student vykonal odbornou praxi a popis pracovního zařazení studenta.
   b) Seznam úkolů zadaných studentovi v průběhu odborné praxe s vyjádřením jejich časové náročnosti.
   c) Zvolený postup řešení zadaných úkolů.
   d) Teoretické a praktické znalosti a dovednosti získané v průběhu studia uplatněné studentem v průběhu odborné praxe.
   e) Znalosti či dovednosti scházející studentovi v průběhu odborné praxe.
   f) Dosažené výsledky v průběhu odborné praxe a její celkové zhodnocení.

Seznam doporučené odborné literatury:

Podle pokynů konzultanta, který vede odbornou praxi studenta.

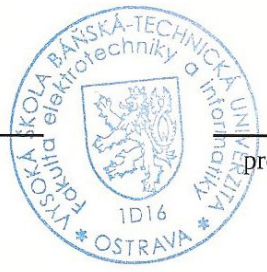
Formální náležitosti a rozsah bakalářské práce stanoví pokyny pro vypracování zveřejněné na webových stránkách fakulty.

Vedoucí bakalářské práce: **Ing. Marek Běhálek, Ph.D.**

Konzultant bakalářské práce: RNDr. Jaroslav Žáček, Ph.D.

Datum zadání: 01.09.2016

Datum odevzdání: 28.04.2017




doc. Dr. Ing. Eduard Sojka
*vedoucí katedry*

prof. RNDr. Václav Snášel, CSc.
*děkan fakulty*

Prohlašuji, že jsem tuto diplomovou práci vypracoval samostatně. Uvedl jsem všechny literární prameny a publikace, ze kterých jsem čerpal.

V Ostravě 7. dubna 2017

.....
podpis studenta

Souhlasím se zveřejněním této diplomové práce dle požadavků čl. 26, odst. 9 Studijního a zkušebního řádu pro studium v bakalářských programech VŠB-TU Ostrava.

V Ostravě 7. dubna 2017

SDE Software Solutions s.r.o.
podpis zástupce
IČ: 253 09 978
tel.: 545 211 280
(3)

**Abstrakt**

V mé baklářské práci, která byla vedena formou praxe, se budu zaobírat kompletním vývojem softwaru pro americké nemocnice. Praxe byla realizována ve firmě Software Development Europe, s. r. o., ve které pracuji na pozici softwarového inženýra. Náplní mé práce a tedy i zadáním bakalářské praxe je vývoj informačního systému pro objednávání pacientů do nemocnic, který je psán v jazycích Ruby on Rails, JavaScript a mnohých dalších.

**Klíčová slova**: Odborná praxe, Ruby on Rails, informační systém, nemocnice, webová aplikace

**Abstract**

In my bachelor's thesis, that was realized by individual professional practice in the company, I will describe complete software development for American hospitals. The practice was realized in a company Software Development Europe, s. r. o., which I am employed in as a software engineer. I am in charge of development of a medical information system that is used for patient scheduling and ordering. The system is written in Ruby on Rails, Javascript and many others.

**Key Words**: Individual Professional Practice in the Company, Ruby on Rails, information system, hospitals, web application

# Obsah

**Seznam použitých zkratek a symbolů** **7**

**Seznam obrázků** **8**

**1 Úvod** **10**
1.1 Historie firmy . . . 10
1.2 O firmě . . . 10
1.3 O projektu . . . 10
1.4 Hierarchie projektu . . . 11
1.5 Technologie . . . 11

**2 Technologie a komunikace** **12**
2.1 Technologie . . . 12
2.2 Komunikace . . . 13

**3 Náplň práce** **16**
3.1 Vývojová metodika . . . 16
3.2 Odeslání SMS pacientovi o nadcházejícím vyšetření . . . 17
3.3 HL7 příchozí a odchozích data . . . 19
3.4 Mapování ICD-9 na ICD-10 . . . 23
3.5 Periodicky se opakující vyšetření . . . 26
3.6 Další vypracované úkoly . . . 27

**4 Získané znalosti** **29**
4.1 Využité znalosti . . . 29
4.2 Scházející znalosti . . . 29
4.3 Nabyté znalosti . . . 29

**5 Závěr** **30**

**Literatura** **31**

# Seznam použitých zkratek a symbolů

| | | |
|---|---|---|
| ICD | – | International Classification of Diseases |
| SDE | – | Firma Software Development Europe |
| AJAX | – | Asynchronous JavaScript and XML |
| Haml | – | HTML abstraction markup language |
| CSS | – | Cascading Style Sheets |
| TDD | – | Test Driven Development |
| DB | – | Database |
| HL7 | – | Health Level-7 |
| VPN | – | Virtual Private Network |

# Seznam obrázků


1 Porovnání rychlostí jednotlivých technologií(méně je lépe)[2] . . . . . 13
2 Popis komunikace mezi jednotlivými systémy . . . . . 14
3 Deployment diagram popisující komunikaci mezi systémy . . . . . 15
4 Okno pro odeslání SMS zprávy pacientovi . . . . . 20
5 Zjednodušený třídní diagram pro tento úkol . . . . . 20
6 Původní mockup zákazníka . . . . . 24
7 Vypracovaná asociace mezi ICD-9 a ICD-10 . . . . . 26
8 Vypracovaní periodicky se opakujících objednávek . . . . . 28

# Seznam výpisů zdrojového kódu

1 Vytvoření tabulky logující odchozí SMS . . . . . 19
2 Parsování dat do formátu JSON . . . . . 22
3 Zpracování příchozí JSON zprávy . . . . . 22
4 Metoda pro odeslání zprávy na centrální server . . . . . 23
5 AJAX požadavek a jeho vyřízení na straně serveru . . . . . 25

# 1 Úvod

V rámci mé individuální odborné praxe se budu zaobírat vývojem užitečné aplikace pro objednávání pacientů do amerických nemocnic. Aplikace umožňuje pacienty objednávat, inteligentně doporučovat vyšetření, ukazovat míru radiace při vyšetření rentgenem a usnadňuje finanční vyúčtování. Odborná praxe probíhala ve firmě Software Development Europe, s. r. o, která je vlastněna dvěma americkými investory, kde jsem již po dobu necelých dvou let zaměstnán jako softwarový inženýr.

## 1.1 Historie firmy

Firma byla založena v Brně v roce 1995 Jeffem Smithem. Jeff Smith si uvědomoval potenciál informačních technologií, a proto se dlouhou dobu zabýval problémem, kde vlastně společnost založit. Pan Smith usoudil, že USA jsou příliš drahé na to, aby se zde založila programátorská firma, a tak se vydal hledat vhodné místo. Navštívil stovky míst, primárně postkomunistické státy, včetně samotného Ruska, ale nejvíce se mu zamlouvalo Brno. Jako hlavní důvody uvádí - způsob smýšlení je blízko západnímu, časový posun proti Severní Karolíně je +6h, úroveň anglického jazyka je nejlepší, ze všech navštívených státu a robustnost programátorů. Jeff Smith dlouhou dobu vedl společnost SDE, avšak v roce 2013 odchází do důchodu a prodává společnost dvěma americkým investorům - Donnie Goinsovi a Miku Millerovi, kteří vedou společnost dodnes.[1]

## 1.2 O firmě

Společnost Software Development Europe, dále jen SDE, je firma střední velikosti a v současné době má 3 pobočky - dvě v ČR a jednu v USA a primárně se zabývá vývojem softwaru a co-sourcingem[1]. Firma má v ČR přibližně okolo 90-ti zaměstnanců - 20 v Ostravě a 70 v Brně. SDE se momentálně soustřeďuje spíše na větší společnosti, případně nadějné startupy. Firma SDE se snaží úzce spolupracovat s univerzitami ve všech směrech, jelikož si je vědoma potencionálu nadějných studentů. Momentální zastoupení studentů na ostravské pobočce je přibližně 40 %. Vize firmy je nabírat motivované jedince s vysokým důrazem na jazykové a komunikační dovednosti a současná úspěšnost přijetí do firmy se pohybuje kolem 4 %. Hlavní důvod, proč si zákazníci vybírají právě nás je kvalita práce a komunikační dovednosti.[1]

## 1.3 O projektu

Projekt, na kterém jsem vykonával bakalářskou praxi se nazýva **iOrder**. Projekt již běží přibližně 8 let a jeho autorem, je pan Tom Miller, který kdysi vytvořil prototyp v PHP. iOrder byl po dobu šesti let implementován pouze jedním programátorem, a to Robem Morrisem, který položil

[1] Bilaterální byznys dohoda, která zaručuje, že práce bude vykonávána jak interním personálem firmy A, tak i externími kontraktory firmy B

tomuto projektu základní kostru a architekturu v Ruby on Rails, kterou nadále rozšiřujeme. Původní záměr projektu byl pouze zefektivnit využití rentgenu v nemocnicích, avšak postupně se projekt rozrostl, a tak momentálně obsahuje i objednávání pacientů do nemocnic, doporučování procedur a mnohem více.

## 1.4 Hierarchie projektu

Jak již bylo zmíněno, SDE je firma zabývající se i co-sourcingem, a tak hierarchie projektu vypadá následnovně. Firma **Novarad**, jež je vlastníkem produktu **iOrder** za náš ostravský tým platí společnosti SDE, tj. veškerý zdrojový kód je vlastnictvím společnosti Novarad a firma SDE slouží jako jakýsi prostředník, který nám zprostředkovává prací a vyplácí nám finanční kompenzaci za naši práci. V Ostravě jsme na projektu momentálně dva programátoři, avšak se počítá s rozšířením našeho týmu v druhém kvartálu tohoto roku. Společnost Novarad má také svůj tým lidí, kteří pracují na projektu iOrder a se kterými přímo komunikujeme, jak verbálně, tak neverbálně formou emailů. Novarad iOrder tým nejsou programátoři ani se nezabývají vývojem či implementací softwaru, jsou v roli zákazníka - product owner[2].

## 1.5 Technologie

Na našem projektu používáme široké spektrum technologií, avšak v mé bakalářské práci se budu soustředit pouze na ty nejdůležitější z nich - Ruby on Rails, jQuery, CSS a PostgreSQL a Ruby on Rails frameworky ironcore, irongaze od původního programátora projektu - Roba Morrise.

[2]Vlastník produktu, reprezentuje zainteresované subjekty a je hlasem zákazníka. Je zodpovědný za to, že tým do byznysu přidá hodnotu

# 2 Technologie a komunikace

## 2.1 Technologie

V této sekci se budeme zabývat použitými technologiemi s důrazem na Ruby on Rails.

### 2.1.1 Ruby

Jazyk Ruby byl vyvinut Yukihirem Matsumotem v Japonsku a jeho prvotní verze byla dostupná v roce 1995. Ruby vychází z jazyků Perl, Smalltalk, Eiffel, Ada a Lisp. Tento jazyk podporuje více programovacích paradigmat, včetně funkčního, objektového a imperativního programování. Náš projekt je postaven na verzi 2.1.5. Využívá gemů, což je vlastně jiné označení pro pluginy. [6]

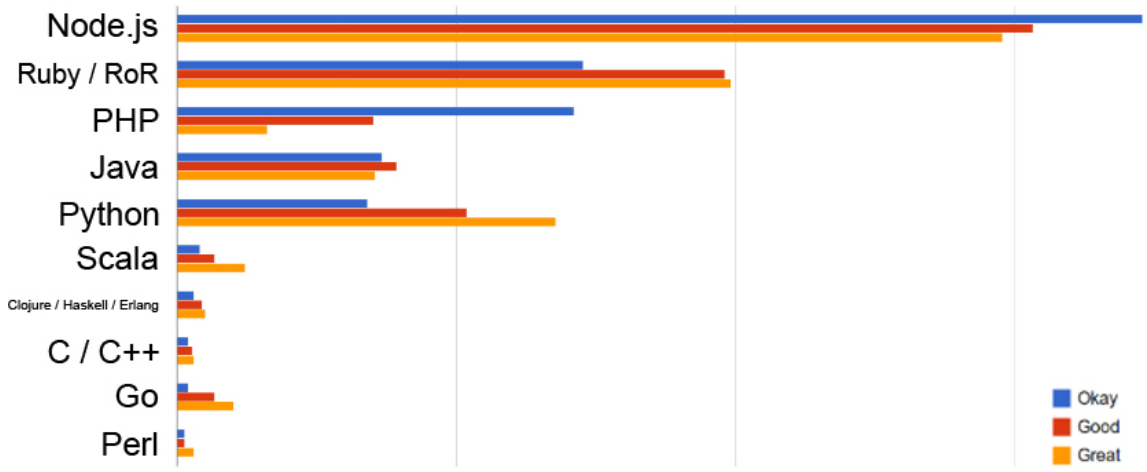
### 2.1.2 Ruby on Rails

Ruby on Rails je framework, postaven na jazyce Ruby, pro vývoj webových aplikací, jež využívá architekturu model-view-controller a byl vytvořen Davidem Hanssonem a běží pod licencí MIT. Využívá integrovaného objektově-relačního mapování, které se jmenuje ActiveRecord. Využíváme verzi Ruby on Rails 3.0.1, ovšem se jedná o nefunkční požadavek zákazníka, takže momentální upgrade na aktuálnější verzi Ruby on Rails možný není. [4]

### 2.1.3 Proč Ruby on Rails?

Zajisté existují rychlejší jazyky a frameworky než Ruby on Rails, např. **ASP.NET MVC**. Proč se tedy většina momentálních startupů, nejenom v Sillicon Valley, rozhodla zvolit právě Ruby on Rails? Důvodu je hned několik[3][4]:

- Velice rychlý vývoj aplikace, jelikož například ORM je již součástí frameworku
- Intuitivní a jednoduchá syntaxe
- Poměrně velké množství vývojářů a designérů preferují použití Haml a Sass CSS preprocesoru, jež jsou obojí součástí Ruby on Rails
- Je Open source
- Má skvělou dokumentaci a komunitu
- Možnost definovat datový model přímo skrze Ruby kód
- Podpora unit testů - je tedy možné zvolit TDD
- Z pohledu zákazníka, firmy jež staví aplikace na frameworku Ruby on Rails působí profesionálněji z důvodu využívání nových moderních technologií




Obrázek 1: Porovnání rychlostí jednotlivých technologií(méně je lépe)[2]

### 2.1.4 Databáze

Jako systém řízení báze dat byl zvolen PostgreSQL, jež je jeden z nejpoužívanější druhů databáze pro Ruby on Rails aplikace, a to hlavně z důvodu lepší podpory ActiveRecord, avšak alternativou by byla databáze MySQL, která ovšem nenabízí práci s poli a outter join a nemá tak dobrou dokumentaci jako PostgreSQL. Databáze má v současnosti okolo 110 tabulek, takže projekt je opravdu komplexní.

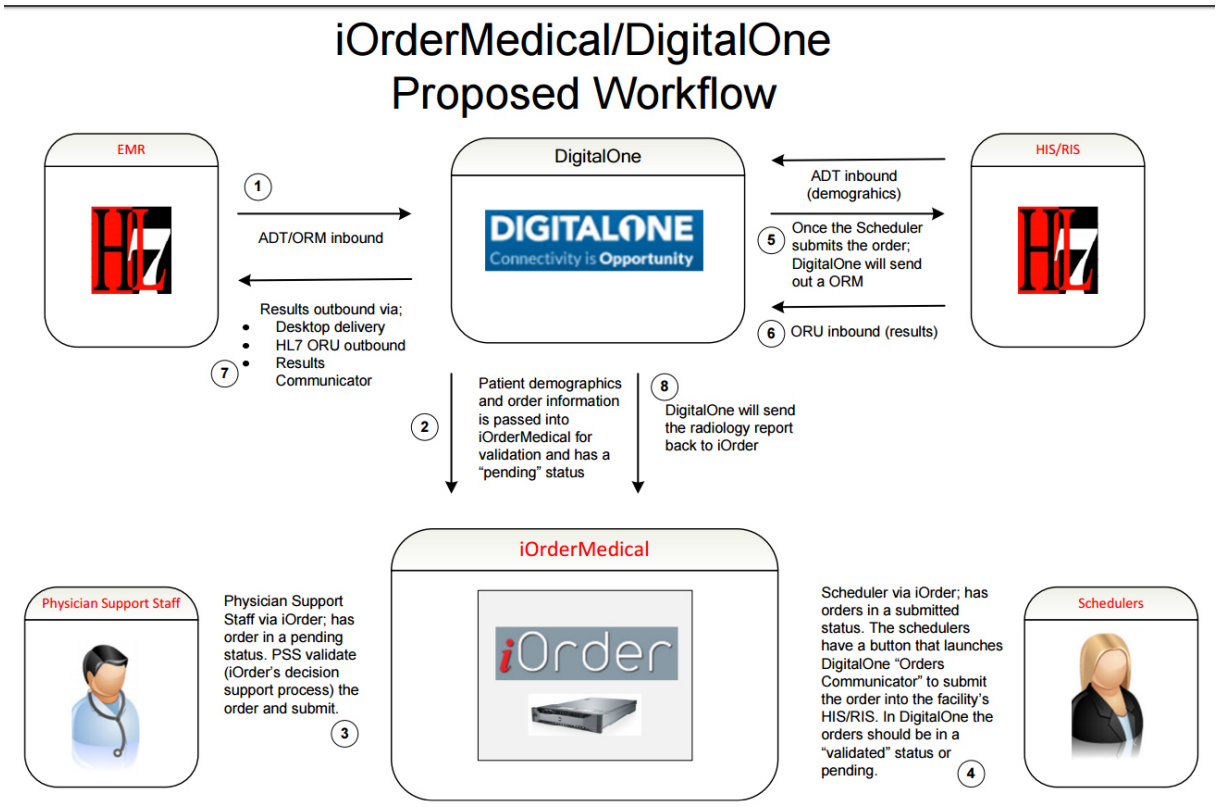
### 2.1.5 HL7

HL7 neboli Health Level 7 je mezinárodní standard pro transfér dat pacientů mezi aplikacemi[7]. Dále obsahuje engine, pro překlad dat z formátu HL7 na vybraný formát. V našem případě JSON.

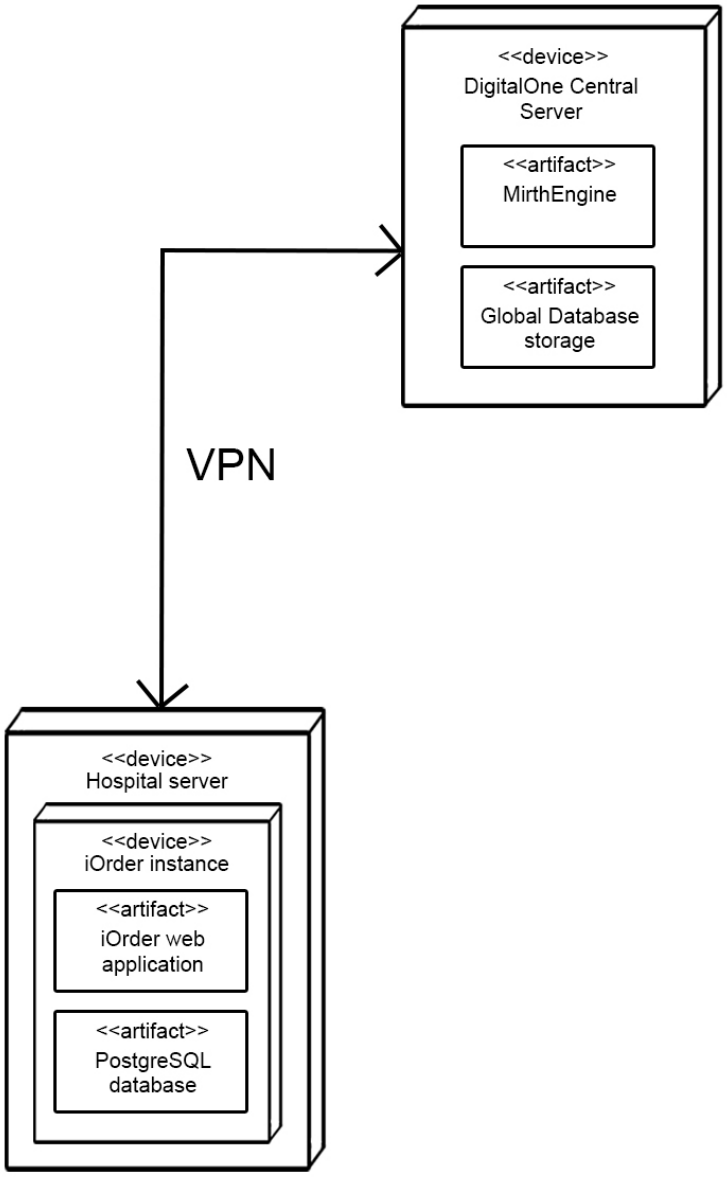
## 2.2 Komunikace

Abychom mohli pokračovat dále bude potřeba si projít základní schéma a principy projektu iOrder. Jak již bylo zmíněno, iOrder je software primárně na objednávání pacientů do nemocnic na vyšetření. Tedy iOrder beží na úrovni nemocnic. Každá nemocnice tedy má svou instaci hostovanou na svém vlastním serveru, který je umístěn fyzicky v nemocnici. K tomuto systému mohou přistupovat pouze skrz webové rozhraní, tedy pouze přes protokol HTTPS. Každá nemocnice je připojena do sítě VPN, která vede na centrální server společnosti DigitalOne. Do centrálního serveru proudí data z dalších zdrojů skrze protokol HL7 viz výše. Tato data jsou přeložena z HL7 formátu do formátu JSON a dále odesílána skrz VPN do jednotlivých instancí iOrderu, tj. do nemocnic.

Náš projekt iOrder se tedy bude zabývat jen informačním systémem nemocnic, tedy systémem iOrder.




Obrázek 2: Popis komunikace mezi jednotlivými systémy




Obrázek 3: Deployment diagram popisující komunikaci mezi systémy

# 3 Náplň práce

Náplní mé práce, jako softwarového inženýra, bylo implementovat systém iOrder, komunikace se zákazníkem v anglickém jazyce, odhadování pracnosti úkolů, navrhování architektury a optimalizace kódu. Úkolů za dobu mé bakalářské praxe jsem vykonal desítky, avšak jsem se rozhodl popsat jen netriviální.

## 3.1 Vývojová metodika

Náš projekt je v současnosti založen na agilním vývoji metodikou Scrum, což v našem případě znamená, že Sprint, tj. období, ve kterém musí byt splněny všechny zadané úkoly, trvá měsíc. Obecně by Sprint neměl být delší než 21 dní, jelikož tato metodika ztrácí smysl, avšak tento způsob byl zvolen partnerskou co-sourcingovou firmou Novarad a projektový manažer je s touto metodikou spokojen.

### 3.1.1 Průběh cyklu

Těsně před začátkem Sprintu nám náš iOrder USA tým zašle seznam úkolů, které bude v brzké době třeba udělat. Já a můj spolupracovník, který nastoupil společně se mnou, odhadneme metodikou poker planning, kolik nám jednotlivé úkoly zaberou času a náš odhad pošleme našemu USA týmu vyplněný. V případě dotazu se obrátíme na USA tým s přesným dotazem, co chceme upřesnit. USA tým si naše odhady projde a podle priorit a našeho odhadu vybere úkoly pro nadcházející Sprint.

Na začátku Sprintu nám projektový manažer představí úkoly, na kterých máme tento Sprint pracovat a my si s kolegou interně práci rozdělíme. Naši práci průběžně ukládáme skrz verzovací systém Git na server Github do privátního repozitáře, který je vlastněn společností Novarad - jak již jsem zmiňoval, v případě našeho projektu společnost SDE vystupuje jako co-sourcingová firma, a tak veškerý kód patří partnerské společnosti.

Po provedení určitého úkolu si navzájem s kolegou provádíme code review (je-li čas), případně při složitých úkolech programujeme formou párového programování, kdy jedna osoba píše kód a druhá tento kód rovnou kontroluje.

### 3.1.2 Komunikace se zákazníkem

S naším partnerským USA týmem obvykle verbálně komunikujeme 1-2 týdně formou meetingů, které obvykle zaberou 30 minut a jsou v anglickém jazyce. Během meetingů se disktuje aktuální průběh Sprintu, případné problémy s úkoly či problémy s lidmi, dozvíme se feedback od amerických nemocnic, vlastnící náš produkt a mnohé další. Neverbální komunikace je takřka na denním pořádku, a taktéž probíhá v jazyce anglickém.

### 3.1.3 Konec cyklu

Na konci Sprintu se vytvoří nová verze systému, která bude dostupná pro všechny nemocnice, které využívají náš produkt. Při implementaci složité části systému, provedeme online prezentaci naším kolegům ve firmě Novarad. Pokud úkoly ve Sprintu byly zaměřeny spíše na opravu chyb či implementaci jednoduchých částí systému, pak prezentace není potřebná a stačí vydat dokumentaci.

## 3.2 Odeslání SMS pacientovi o nadcházejícím vyšetření

### 3.2.1 Popis požadavku

Náš USA tým nám sdělil, že by si zákaznící přáli odeslat SMS pacientovi, když nemocnice schválí jeho vyšetření a bude znát jasný termín. Prvotně je třeba najít službu, která dovoluje odesílat SMS do různých sítí, a pak již jen implementovat. Tento úkol se nám zprvu složitý, jelikož jsem neměl absolutní představu, kolik operátorů se v USA nachází. Takže jsem se obrátil na náš USA tým, který tento dotaz zodpověděl a částečne poskytl řešení tím, že si zakoupili službu, která dovoluje odeslat SMS do všech sítí zdarma - stačí odeslat email v následujícím formátu **telefoníČísloPacienta@doménaProvidera**. Dále pak jsem se řídil sloganem firmy, jež zní „Contribute to success“ a navrhl jsem, že by bylo dobré uchovávat logy o odeslaných SMS zprávách, umožnit odesílání SMS automaticky i manuálně a možnost nastavit formát odchozí zprávy, a také jsem přidal SMS, která pacienta notifikovala, když se vyšetření zrušilo. Tento úkol byl estimován na přibližně 100 hodin.

### 3.2.2 Řešení

Je logické, že není možné dělat vše najednou, a tak je třeba systematicky rozebrat úkol do jednotlivých dílčích úkolů. Rozhodli jsme se, že já vypracuji vše, kromě možnosti upravit výchozí zprávu do libovolného formátu - tuto dílčí část si převzal kolega. Co je tedy potřeba udělat?

- Vytvořit mailer, který bude odesílat emaily v daném formátu
- Vytvořit novou databázovou tabulku, která ponese informace o odeslaných SMS zprávách
- Vytvořit všechny požadované view a akce v controllerech
- Přidat tlačítko na manuální odeslání SMS zprávy a indikátor, že SMS byla odeslána
- Vytvořit checkbox, kterým budu ovládat, zda-li se mají SMS zprávy odesílat automaticky po objednání pacienta či nikoliv
- Vytvořit tabulku **sms_providers**, která ponese dostupné providery
- Vytvořit selectbox u pacienta, kterým budeme volit jeho providera, jelikož momentálně neexistuje deterministický způsob určení providera pouze na základě telefoního čísla

- Vytvořit u pacienta další atribut s názvem **sms_phone_number**, jelikož v USA jsou stále v módě klasické drátové telefony
- Vytvořit u pacienta referenci na providera

### 3.2.3 Tvorba maileru

**OrderMailer** bude naše hlavní třída, která bude volána při vzniku jakékoliv odchozí SMS zprávy. Rails už v základu obsahuje třídu **ActionMailer::Base**, která definuje funkci **mail(to,subject)**. Dále pak tato funkce pracuje s hodnotami třídních proměnných **@content** a **@from**. Nadefinujeme-li tyto proměnné v naší třídě, která dědí právě z **ActionMailer::Base**, může funkce **mail** i **deliver** s hodnotami těchto proměnných pracovat. Vytvoříme si tedy v naší třídě OrderMailer metody **scheduled_mail(order,content = nil, number = nil, provider = nil)** a **cancellation_mail(order,content = nil**. Jak je již z názvu patrné, tak jedna metoda bude volána, když bude pacient na vyšetření objednán a druhá, když bude vyšetření zrušeno.

Možná jste si všimli, že metody mají jiné parametry a přitom dělají téměř shodnou věc. Ano i ne. Obě metody budou volány automaticky - metoda cancellation_mail bude volána automaticky vždy při zrušení objednávky a metoda scheduled_mail bude volána automaticky jen tehdy, zda-li nemocnice má povolenou možnost, aby se SMS zprávy odesílaly automaticky po objednání pacienta na vyšetření. Rozdíl je ovšem v tom, že metoda scheduled_mail má i manuální režim, tedy je možné obsah zprávy editovat i pro jednu specifickou objednávku a tak je potřeba data z formuláře předat do metody.

### 3.2.4 Tvorba databázových tabulek

Tvorba tabulek se v Ruby on Rails dělá přes migrace, které jsou psány v Ruby kódu, což je skvělý přístup, protože můžeme danou migraci kdykoliv přepsat a vše je přehledně uloženo v jednom adresáři. Samozřejmě je možné databázový model vytvářet pomocí různých nástrojů, ale standardem jsou migrace. Bude tedy potřeba rozšířit tabulku *pacient* o atributy sms_phone_number a referenci na *provider* tabulku, kterou musíme taky vytvořit. Dále pak vytvoříme tabulku *sms_loggers*, která je zapsaná pomocí Ruby kódu níže. U této tabulky stojí za zmínku atribut **number**. V našem databázovém schématu sice tabulka *orders* obsahuje referenci na pacienta, tj. máme přímo přístup k jeho telefonímu číslu, ale to se může změnit, a tak je třeba uchovávat skutečně číslo, na které byla zpráva odeslána.

```ruby
class SmsLoggers < ActiveRecord::Migration

  def change
    create_table(:sms_loggers) do |t|
      t.belongs_to :order
      t.string 'number'
      t.string 'provider'
      t.string 'content'
      t.boolean 'sent', default: false
    end
  end

end
```

Výpis 1: Vytvoření tabulky logující odchozí SMS

Teď když již máme tabulku **sms_loggers** můžeme vytvořit její model, do kterého zatím připíšeme referenci na order a naopak do modelu Order připíšeme vazbu 1:N, protože se může stát, že se SMS zpráva zašle mnohokrát.

### 3.2.5 Dokončení úkolu

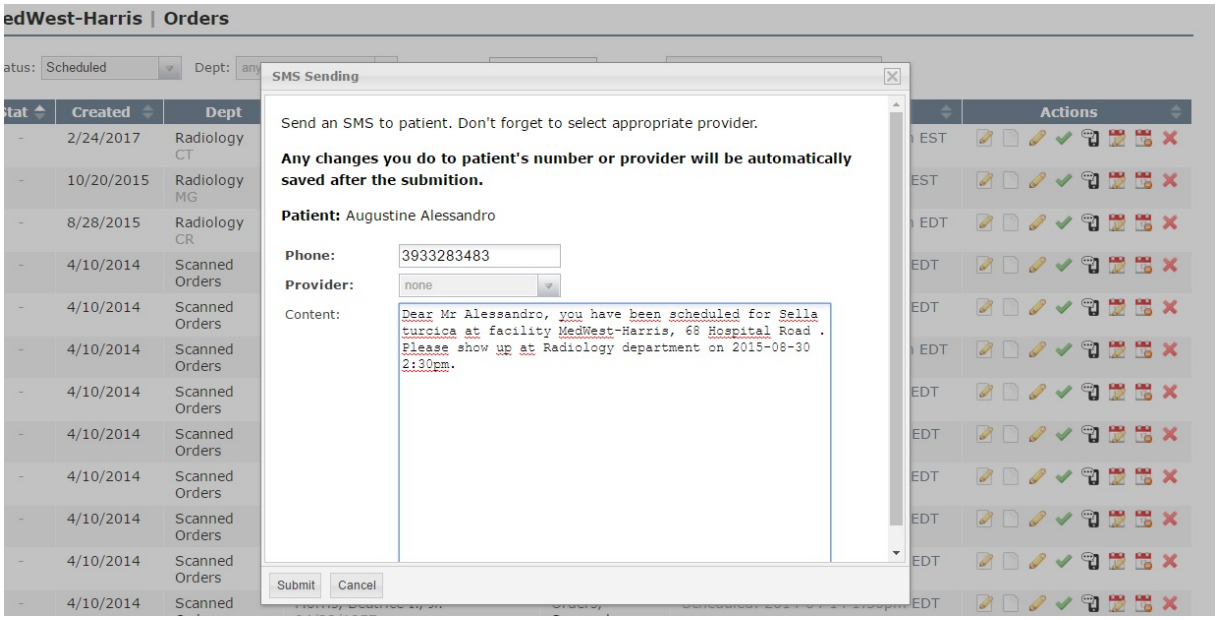
Po vytvoření všech požadovaných tabulek, modelů, controllerů a view se dostávame do konečné fáze. Jako rozhraní jsem zvolil pop-up okno, které se zobrazí na popředí v případě manuálního odesílání zpráv. Text se tedy předvyplní na základě výchozí šablony, kterou si mohou jednotlivé nemocnice upravovat. Metoda scheduled_mail je tedy volána v případě objednání pacienta na vyšetření v daném termínu a metoda cancellation_mail je volána při zrušení objednávky. V obou případech jsou SMS zprávy zaznamenány do tabulky logů.

## 3.3 HL7 příchozí a odchozích data

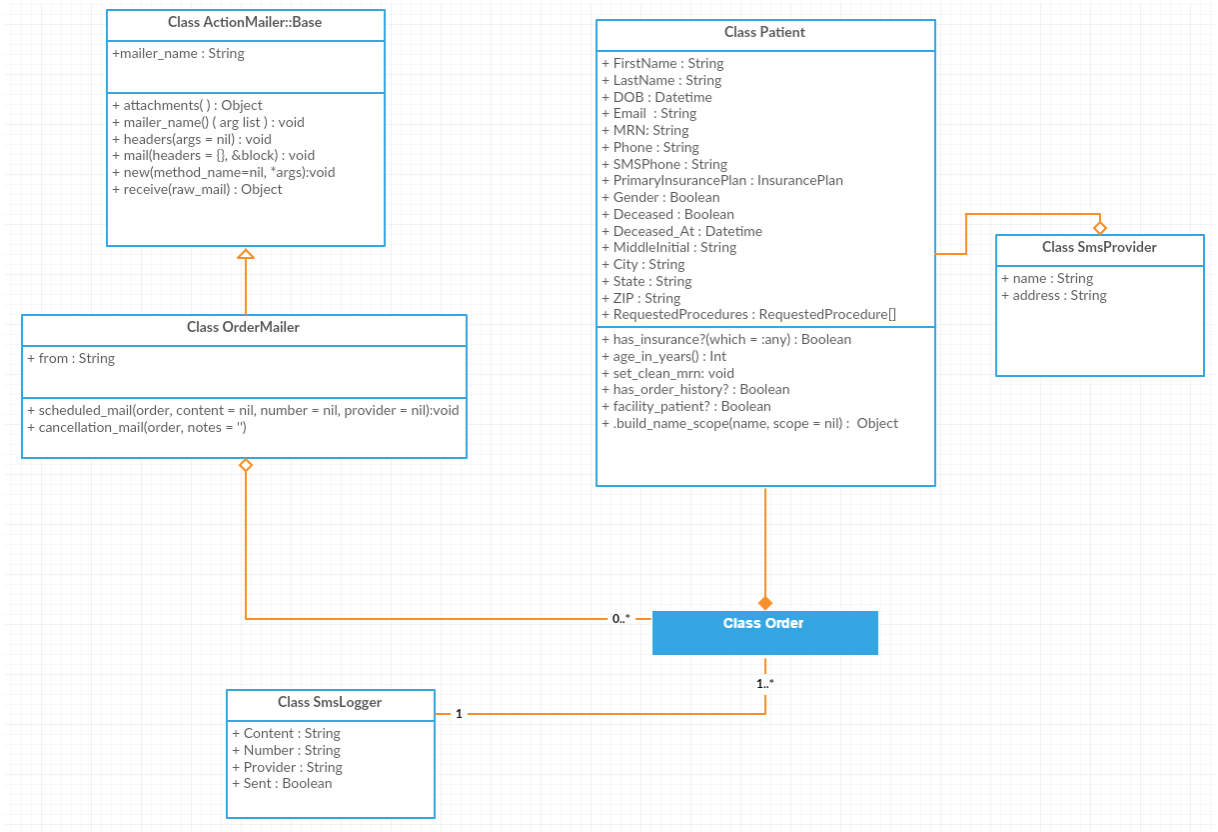
### 3.3.1 Popis požadavku

Jeden z hlavních účelů iOrder systému je ulehčovat práci s daty a umožňovat kontrolu předpokladů potřebných k zjištění, zda-li je pacient schopen zdravotní vyšetření či léčbu schopen podstoupit či nikoliv. Jistě si uvědomujete pracnost, při vytváření pacienta v systému či vytváření jeho zdravotní historie. Od toho je tady centrální systém, který je schopen rozposílat data ostatním serverům. Přijde-li pacient na vyšetření do nemocnice A, pak jeho zdravotní osobní data obdrží centrální server. Tento server je schopen data dále distribuovat ostatním instancím.

Úkolem bylo naimplementovat schopnost přijímat a odesílat data pacientů. Určili jsme si formát zasílání dat a požádali jsme inženýry pracující na centrálním serveru o odesílání v tomto formátu. Jak již je zvykem čím více lidí, tím větší byrokracie a prostor pro chyby. Inženýři pracující na centrálním serveru byli schopni až po týdnu dodat požadovány formát zprávy, takže jsem tento čas trávil opravami různých chyb.




Obrázek 4: Okno pro odeslání SMS zprávy pacientovi




Obrázek 5: Zjednodušený třídní diagram pro tento úkol

Tento úkol byl odhadnut na 200 hodin, a to hlavně z důvodu blízké spolupráce s inženýry z centrálního systému, kteří měli defacto stejnou funkcionalitu implementovat na centrálním serveru.

### 3.3.2 Řešení

Úkol jsem jako vždy rozdělil do podúkolů, protože požadavek zákazníka zněl doslova „Je potřeba vytvořit spojení s centrálním serverem a odesílat a přijímat data“. Jako způsob přenosu dat jsem se rozhodl použít JSON a to z důvodu malé velikosti souboru.

- Vytvořit formát pro posíláni objednávek a dat pacientů
- Schopnost přijímat objednávky
- Schopnost přijímat data pacientů
- Schopnost odesílat objednávky
- Schopnost odesílat data pacientů
- Vytvořit GUI rozhraní pro přijímané objednávky

### 3.3.3 Schopnost přijímat data pacientů a objednávky

Jak již jsem zmiňoval v sekci komunikace, tak mezi centrálním server a každou instancí je vytvořen tunel pomocí technologie VPN, data by tedy měla být při přenosu teoreticky chráněná. Přenos dat tedy probíhá přes HTTP.

Pro příjem zpráv jsem si definoval speciální URL, na kterou budou data zasílat. Data, která budou na tuto adresu odeslána budou prvotně zapsána do souboru, jelikož by se mohlo stát, že by se obsah stránky v průběhu zpracováni změnil. Soubor bude uložen na server s příponou .json. Dále pak pracuji jen s tímto souborem.

Ruby on Rails poskytuje relativně pohodlné parsováni formátu JSON, takže po použití funkce parse_json se již data budou nacházet ve formátu JSON.

```
  def parse_json(json)
    if json.blank?
      fail ('Missing JSON data')
    else
      json = JSON.parse(CGI::parse(json).to_json)
      if json.blank?
        fail ('Unable to parse JSON data')
      else
        return json
      end
    end
    return nil
  end
```

Výpis 2: Parsování dat do formátu JSON

V případě chyby volám metodu fail, která notifikuje centrální server, že něco není v pořádku.

```
def process_message(version, json)

    //Check version info
    unless version == 'v1'
      fail ('Unsupported API version: ' + version.inspect)
      return
    end

    //Get incoming JSON data
    json = parse_json(json)
    return if json.blank?

    //Parse out top-level message fields
    @message_id = !json['msg_id'].blank? ? json['msg_id'][0] : ''
    @timestamp = parse_timestamp(json['msg_timestamp'].blank? ? json['msg_timestamp'][0] : '')
    @message_type = json['type'][0]
    return unless find_source(json['source'][0], json['integration_key'][0])

    //Check for patient info
    if patient_data = json['type'][0] == 'patient'
      //We have new demographics info - update/create record
      return unless create_patient(json)
    end

    if proc_data = json['type'][0] == 'procedure'
      return unless create_patient(json)
      return unless add_requested_procedure(json)
    end
end
```

Výpis 3: Zpracování příchozí JSON zprávy

Z kódu je vidět, že v případě že přijdou data pacienta, pak pouze vytvoříme či aktualizujeme záznam v tabulce, avšak v případě vytvoření objednávky musíme pacienta vytvořit či aktualizovat stejně. To je velice logické, jelikož tabulka order obsahuje referenci na pacienta, takže prvně vytvoříme pacienta, a až pak mu přiřadíme vyšetření.

### 3.3.4 Schopnost odesílat data pacientů a objednávky vyšetření pacienta

V opačném případě chceme posílat veškerá data na centrální server. Zde se bude jednat pouze o zaslání jedné zprávy, tj. zpráva bude obsahovat jak data pacienta, tak i proceduru, kterou podstoupí.

Data budou odeslána v případě, že objednávka na vyšetření bude ve stavu **submitted** - tento stav identifikuje objednávku, která neobsahuje žádné chyby a má vyplněny veškeré náležitosti.

Pro odesílání dat na centrální server, slouží primárně metoda **post_message(msg)**, která nedělá nic jiného, než že vezme naši zprávu v formátu JSON a odešle ji pomocí HTTP requestu na námi zvolený **ENDPOINT**, což je vlastně IP adresa centrálního serveru.

```
def post_message(msg)
  uri = URI.parse(ENDPOINT)
  Net::HTTP.new(uri.host, uri.port).start do |client|
    request                  = Net::HTTP::Post.new(uri.path)
    request.body             = msg.to_json
    request["Content-Type"] = "application/json"
    res = client.request(request)
  end
end
```

Výpis 4: Metoda pro odeslání zprávy na centrální server

### 3.3.5 Dokončení úkolu

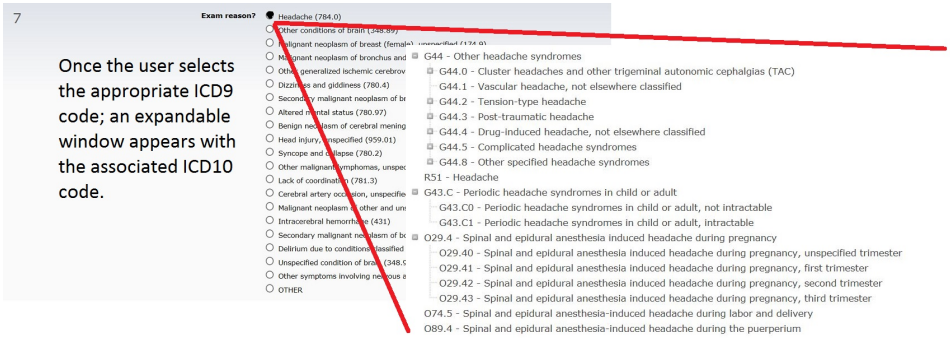
I přes původní odhad 200 hodin tento úkol zabral podstatně více, a to z důvodu pomalé implementace na straně centrálního serveru.

## 3.4 Mapování ICD-9 na ICD-10

### 3.4.1 Popis požadavku

Jeden z hlavních a klíčových požadavků byla implementace funkce, která měla zajišťovat doktorům a nemocničnímu personálu využívání zaběhnutého systému ICD-9, který obsahuje kódová označení pro různé druhy nemocí. Tato verze obsahuje 13000 kódů, tj. nemocí. Avšak nemocnice jsou nuceny používat verzi ICD-10, která obsahuje 68000 nemocí[5]. Verze ICD-10 je relativně nová, a tak personál není na tuto verzi zcela zvyklý. Vznikl tedy požadavek, který umožní personálu pořád používat systém ICD-9, avšak po zvolení kódů ICD-9 vyskočí asociována tabulka ICD-10 kódu. Je tedy patrné, že tento způsob bude doporučovat ICD-10 kódy na základě kódu

ICD-9, a že jednomu ICD-9 kódu bude nejspíše odpovídat více ICD-10 kódů, jelikož ICD-10 kódů je přibližně pětinásobně více než ICD-9 kódů[5].




Obrázek 6: Původní mockup zákazníka

Tento úkol byl odhadnut na 140 hodin, a to hlavně z důvodu neznalosti konverze mezi ICD-9 a ICD-10.

### 3.4.2 Řešení

Tento úkol byl z počátku velice náročný - neměl jsem téměř žádne informace o mapování. Psal jsem e-mail organizaci www.cms.gov, která se správou těchto kódů zabývá a ti mi poslali několik konverzních tabulek. Tyto tabulky jsem předal našemu týmu v USA, aby je analyzoval a vybral tu nejlepší.

- Zjistit více o konverzích
- Vytvořit datový model a tabulku
- Vytvořit AJAX požadavek, který nám vrátí příslušné ICD-10 kódy po kliku na kód ICD-9
- Vytvořit Javascript funkci, pro zobrazení okna, při zvolení ICD-10

### 3.4.3 AJAX požadavek a Javascript funkce

Po úspěšném vytvoření asociační tabulky a modelu bude potřeba vytvořit příslušné volání metody pro transformování ICD-9 kódu na kód ICD-10. K tomu nám bude sloužit AJAX požadavek, který odešleme do našeho kontroleru s příslušnými informacemi o zvoleném ICD-9 kódu.

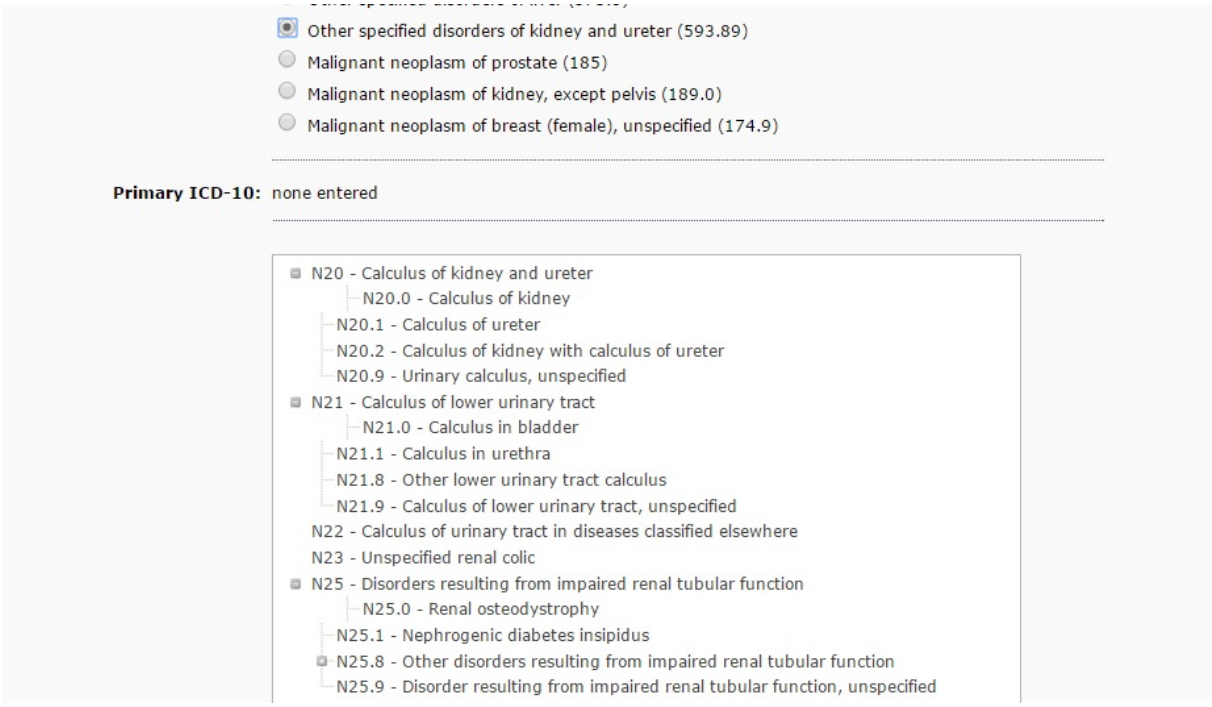
```
 // <Javascript>
Core.postAJAX('/icd10/_transform_icd_code/', params, function (success, data)
 {

  if (success && data.id != null) {
       ...
       //create a tree with provided ICD-10 ids
       ...
  }
  else {

       //do not display tree, no codes found
       tree.attr('style', 'display:none');
  }
});

 // <Ruby>
 def _transform_icd_code
  code = params[:id]
  @return = Icd9Icd10Association.select(:icd_10_code).where(:icd_9_code => code).pluck(:icd_10_code).
       first
    render :json => {
       :ids => @return
   }
end
```

Výpis 5: AJAX požadavek a jeho vyřízení na straně serveru

Jak je vidět, tak si díky našemu AJAX volání získáme veškeré ICD-10 kódy, jež jsou v naší asociační tabulce vázány na specifický ICD-9 kód a tyto ID vracíme zpátky do Javascript metody, odkud si zavoláme metodu pro vykreslení listview, který již bude obsahovat seznam stromové struktury oněch ICD-10 kódů.

### 3.4.4 Dokončení úkolu

Úkol mi zabral 120 hodin, což se blíží času, který jsme odhadli.




Obrázek 7: Vypracovaná asociace mezi ICD-9 a ICD-10

## 3.5 Periodicky se opakující vyšetření

### 3.5.1 Popis požadavku

Zákazník nás upozornil, že se může vyskytnout vyšetření, které se může opakovat v určitém časovém intervalu, tak je třeba vytvořit funkcionalitu, která nám pacienta objedná za právě námi specifikovaný časový interval. Při vytvoření vyšetření chtějí mít k dispozici tlačítko, které otevře pop-up, kterým budeme mít možnost naši objednávku opakovat v námi zvoleném intervalu - na výběr mají být 4 možnosti: každých A minut, každých B hodin, každých C dní a každých D týdnů. Dále pak má naše okno obsahovat dropdown, který reprezentuje, kolikrát se má naše objednávka na vyšetření znovu vytvořit.

Tento úkol byl odhadnut na 50 hodin.

### 3.5.2 Řešení

Je patrné, že budeme muset naší první objednávku zkopírovat a budeme měnit jen datum odeslání objednávky do nemocnice. Po odeslání objednávky do nemocnice personál objedná pacienta a určí čas průběhu jeho vyšetření. Budeme tedy muset zajistit, aby se nám objednávky, které vznikly díky naší funkcionalitě, nezobrazovaly personálu nemocnice až do doby, kdy aktuální čas nebude větší nebo roven času odeslání objednávky - nechceme aby nám personál objednával pacienta dopředu.

- Vytvořit pop-up s potřebnými elementy

- Vytvořit algoritmus pro výpočet času odeslání všech opakujících se objednávek
- Vytvořit scope, který skryje objednávku, dokud nebude aktuální čas větší nebo roven času odeslání objednávky

Při řešení této úlohy jsem se potýkal s jedním problémem - co se stane, když personál smaže původní objednávku.

### 3.5.3 Smazání původní objednávky

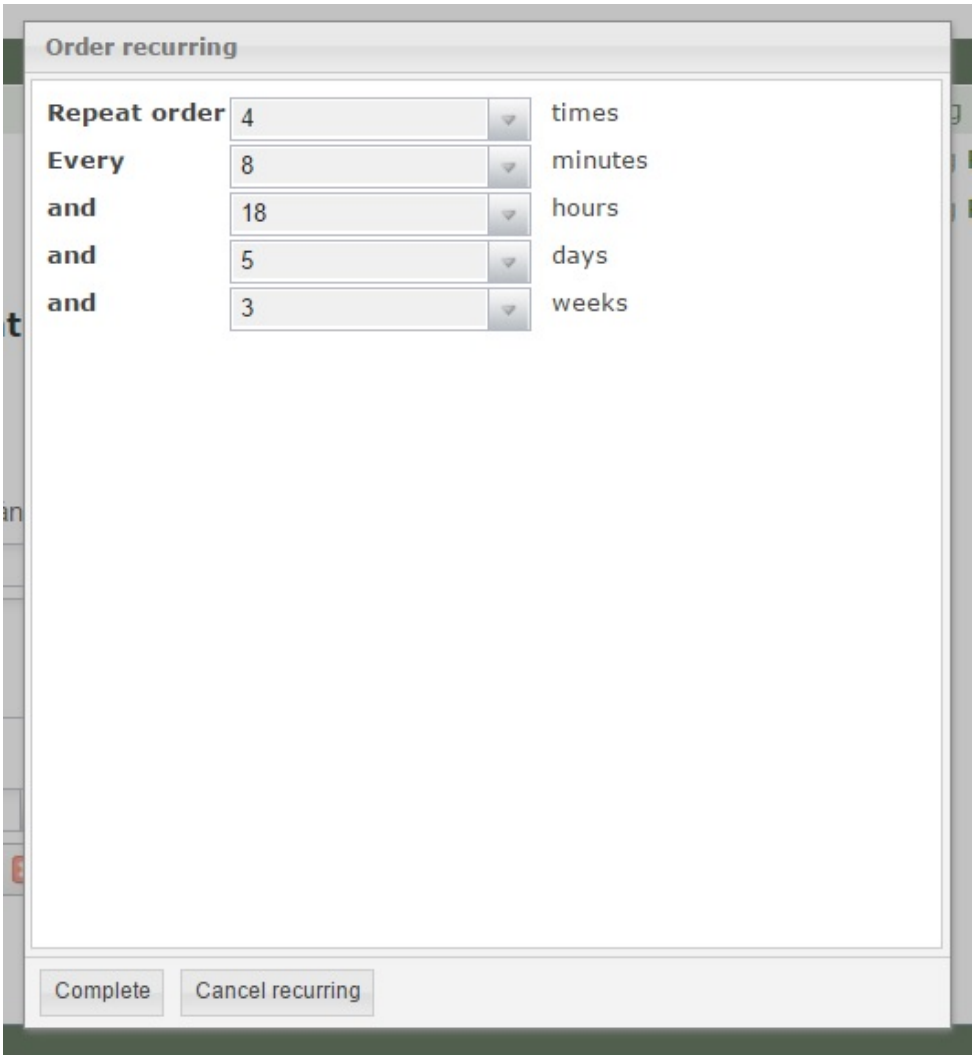
Pokud vytvořím kopii původní objednávky pro naše další opakující se objednávky, tak předám našemu novému objektu všechny vlastnosti a atributy objektu prvního. Objednávka však může obsahovat i přílohu ve formátu PDF či JPEG - vyskytl se tedy problém. Pokud smažu objednávku, pak se smaže i ona příloha. Jedno řešení tedy může být zrušení této závislosti, avšak toto řešení není zcela správné. Další řešení by bylo, vytvořit přílohu znovu - toto určitě taky není zcela správné řešení, ale zbavili bychom se naší závislosti na původní objednávce. Učinil jsem rozhodnutí, že nejlepší bude si u naší přílohy vytvořit atribut, který reprezentuje, kolik objektů je na něm závislých - v případě, že objednávku smažu, sníží se atribut o jedna, a když vytvořím opakující se objednávku, zvýším čítač. Příloha se tedy smaže pouze pokud bude náš čítač roven nule.

### 3.5.4 Dokončení úkolu

Úkol mi zabral přibližně 45 hodin, což odpovídá našemu celkovému odhadu.

## 3.6 Další vypracované úkoly

- Audit logování citlivých informací
- Tvorba generátoru licenčních klíčů pro iOrder v jazyce C++
- Vytvoření mapování ICD kódů na kódy CPT
- Opravy různých chyb




Obrázek 8: Vypracovaní periodicky se opakujících objednávek

# 4 Získané znalosti

Studium na Vysoké škole báňské mi poskytlo solidní základ pro práci softwarového inženýra, avšak nepokrývá veškeré potřebné znalosti k práci na mém projektu. Ve škole jsem si vybíral spíše předměty, které byly zaměřeny na programování desktopových aplikací, takže je logické, že mi škola nemohla poskytnout řádný základ v oblasti webového vývoje.

## 4.1 Využité znalosti

Využil jsem tedy hlavně znalosti získané z databázových systému, vývoje informačních systémů, softwarového inženýrství, teoretické informatiky a tvorby aplikací pro mobilní zařízení. Když budeme mluvit konkrétně, tak největší přínos pro mě měl předmět úvod do databázových systémů, obzvláště z hlediska návrhu databázových systémů. Znalosti získané během výuky tohoto předmětu využívám dnes a denně. Z vývoje informačních systému jsem si odnesl, jak správně vytvořit architekturu, principy objektově orientovaného programování a návrhové vzory, jež také využívám na denním pořádku.

## 4.2 Scházející znalosti

Co mi ovšem chybělo byly znalosti z vývoje webových aplikací - jak na straně klienta, tak na straně serveru. Na druhou stranu teorie je taková, že škola nemůže naučit všechno - každý je zaměřen jinak. Proto jsem velice rád, že jsem mohl pracovat ve firmě SDE, která v kombinaci s VŠB mé znalosti a kompetence rozhodně zdokonalila.

## 4.3 Nabyté znalosti

Když jsem nastoupil do firmy SDE, tak jsem neměl o jazyce Ruby ani ponětí. Za dobu necelých dvou let, jež jsem součástí týmu iOrder, jsem nabyl zkušenosti, se kterými předčím většinu svých spolužáků, a tak mohu práci ve firmě při studiu jenom doporučit. Nejenže se člověk zdokonalí v oblasti IT a získá praxi, ale taky získá nové kontakty, pozná jak se software reálně implementuje a zlepší si komunikační dovednosti.

Během mé praxe ve firmě SDE jsem se naučil programovat v různých známých jazycích, např. jazyce Ruby a frameworku Ruby on Rails, v jazyce Bash, naučil jsem se i pracovat s různými javascriptovými knihovanami, např. jQuery a React. Ovládl jsem i verzovací systém Git. Dále pak jsem si rozšířil obzory při používání CSS a HTML. A v neposlední řadě jsem mohl i prakticky využít anglický jazyk ke konverzaci se zákazníkem a zjistil jsem, jak skutečný praktický vývoj webových aplikací a informačních systémů probíhá.

# 5 Závěr

Jsem velice rád, že firma SDE mi nabídla práci. Během dvou let jsem se byl schopen evolvovat z nekvalifikovaného programátora, kterých z vysokých škol vypustí tisíce, na téměř senior Ruby on Rails programátora.

Byla mi umožněna práce na seriózním projektu, který se v praxi reálně používá, a tak mě netížil pocit, že veškerá práce přijde vniveč. Všechny úkoly, jež nám byly přiděleny zákazníkem jsem úspěšně vypracoval, aktivně jsem se podílel na jejich zlepšeních a upřímně jsem se snažil přispět k celkovému projektu iOrder. Také jsem se stal asertivnějším, jelikož při nástupu do firmy jsem byl velice introvertní a bál jsem se prosadit svůj názor a celkově před lidmi mluvit. Nyní již tento pocit nemám, a naopak jsem nalezl mou silnou stránku - prezentování v anglickém jazyce je pro mě jednodušší a komfortnější než v jazyce českém. Rád bych se ještě zmínil o odhadování pracnosti jednotlivých úkolů. Zpočátku to pro nás bylo velice těžké odhadnout náročnost. Nejspíš z různých úhlů pohledu na věc a nedostatku zkušeností. Ovšem postupně jsme si na to zvykli, a tak momentálně odhadujeme úkoly s mnohem větší přesností a menším rozptylem než na začátku.

Když se k nám do firmy zastaví náš majitel, Donnie Goins, tak nám vždy poskytne zpětnou vazbu od zákazníka. Zatím s námi zákazník reportuje absolutní spokojenost. Velice se jim líbí naše nasazení, pracovitost a podpora - když se vyskytne kritická chyba, tak pracujeme i mimo pracovní dobu, i během dovolené. Dále se jim líbí náš osobitý přístup k produktu, naše návrhy či výhrady, kvalita práce a rychlost. Možná právě proto se zákazník rozhodl náš tým obohatit v dalším kvartálu tohoto roku a tedy rozšířit náš tým o další členy, a tak to vypadá, že na projektu budu pokračovat i nadále.

# Literatura


[1] Veřejné informace o firmě *Software Development Europe, s. r. o* [online]. Dostupné na http://sde.cz dne 2.3.2017

[2] Porovnání rychlostí jazyků [online]. Dostupné na https://codingvc.com/which-technologies-do-startups-use-an-exploration-of-angellist-data dne 13.3.2017

[3] Výhody Ruby on Rails [online]. Dostupné na https://www.toptal.com/ruby-on-rails/after-two-decades-of-programming-i-use-rails dne 12.3.2017

[4] Informace o Ruby on Rails [online]. Dostupné na https://goo.gl/2VfQsC dne 12.3.2017

[5] Informace ohledně verzí ICD [online]. Dostupné na https://www.unitypoint.org/waterloo/filesimages/For%20Providers/ICD9-ICD10-Differences.pdf dne 10.3.2017

[6] Základní informace o Ruby [online]. Dostupné na https://goo.gl/L229O dne 10.3.2017

[7] Informace ohledně HL7 [online]. Dostupné na http://www.hl7.org/ dne 12.3.2017

[8] Tvorba UML diagramů [online]. Dostupné na https://creately.com/ dne 3.4.2017

[9] Informace ohledně Scrumu [online]. Dostupné na https://goo.gl/xnbPr3 dne 3.4.2017

[10] Informace ohledně co-sourcingu [online]. Dostupné na https://goo.gl/qk1HGE dne 3.4.2017